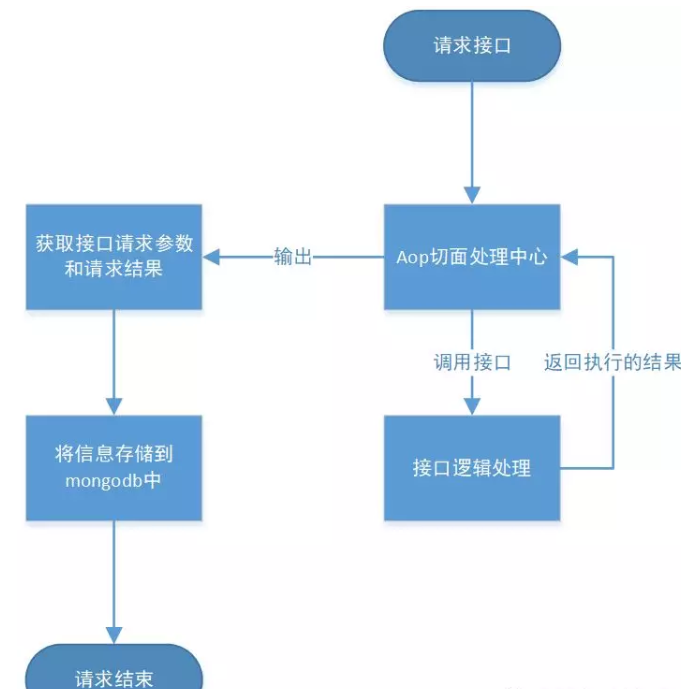
如何通过AOP和自定义注解实现请求日志收集功能



我们先在接口上面添加自定义注解，这样每次请求就都会走AOP的处理中心。在处理中心中，我们可以将请求信息存储到数据库中，方便后期排查问题。

### 自定义注解

我们先来看看自定义注解是如何实现的，代码如下所示：

*/\*\*  
 \* 自定义注解  
 \*  
 \* @author linzhiqiang  
 \* @date 2019/4/26  
 \*/*  
**@Target({ElementType.PARAMETER, ElementType.METHOD})**  
**@Retention(RetentionPolicy.RUNTIME)**  
**@Documented**  
**public** **@interface** RequestLog {  
  
 */\*\*  
 \* 请求模块名称  
 \* @return  
 \*/*  
 **public** String module() **default** "";  
  
 */\*\*  
 \* 接口详情描述  
 \* @return  
 \*/*  
 **public** String operationDesc() **default** "";  
}

这边我们定义了两个字段，一个是接口的模块名称，另外一个就是接口具体的功能描述。

### 接口注释

这两个参数需要在接口处传入，现在我们就来看看接口是如何添加注解和传值的，代码如下所示：

*/\*\*  
 \* request测试专用  
 \* @return  
 \*/*  
 @RequestLog(module = "requestTest", operationDesc = "request测试专用")  
 @RequestMapping(value = "requestTest", method = RequestMethod.POST)  
 public String requestTest(@RequestBody ArticleSubjectDto articleSubjectDto) {  
 String result = null;  
 **try** {  
 System.out.println("我的是方法");  
 result = "请求成功";  
 }catch (Exception e){  
 logger.error("requestTest查询失败", e);  
 return JsonUtils.**to**Json(ResponseUtils.failInServer(result));  
 }  
 return JsonUtils.**to**Json(ResponseUtils.success(result));  
 }

我们可以看到，在接口上面添加我们自定义的注解，然后写上对应的参数值就可以了。

### AOP处理中心

最后我们来看看最核心的AOP处理中心是如何实现的，代码如下所示：

*/\*\*  
 \* 切面AOP  
 \* \*/*  
@Aspect  
@Component  
public **class** SystemLogAspect {  
 **private** static final Logger logger = LoggerFactory.getLogger(SystemLogAspect.class);  
 **private** static final String UNKNOWN = "unknown";  
  
 @Autowired  
 **private** LogRepository logRepository;  
 */\*\*  
 \* 1，表示在哪个类的哪个方法进行切入。配置有切入点表达式。  
 \* 2，对有@SystemLog标记的方法,记录其执行参数及返回结果。  
 \*/*  
 @Pointcut("execution(\* com.minimal..controller..\*.\*(..))&&@annotation(com.minimal.common.sdk.log.RequestLog)")  
 public void controllerAspect() {  
 }  
  
 */\*\*  
 \* 配置controller环绕通知,使用在方法aspect()上注册的切入点  
 \*/*  
 @Around("controllerAspect()")  
 public Object aroundMethod(ProceedingJoinPoint point) throws Throwable {  
 **if** (logger.isDebugEnabled()) {  
 logger.info(">>>>>>>>>>>>>>>进入日志切面<<<<<<<<<<<<<<<<");  
 }  
 *// 获取接口的路径地址*  
 String methodTarget = point.getTarget().getClass().getName() + "." + point.getSignature().getName() + "()";  
 HttpServletRequest request = ((ServletRequestAttributes) RequestContextHolder.getRequestAttributes()).getRequest();  
 OperateLogPO operLogPO = **new** OperateLogPO();  
 operLogPO.setId(UUID.randomUUID().**to**String());  
 operLogPO.setMethod(methodTarget);  
 operLogPO.setCreateTime(new Date());  
 operLogPO.setIp(getClientIpAddr(request));  
 operLogPO.setBrownerNo(getBrownerNo(request));  
 operLogPO.setOsNo(getOsNo(request));  
 *// 获取接口的请求参数*  
 operLogPO.setParams(JsonUtils.toJson(point.getArgs()));  
 MethodSignature signature = (MethodSignature) point.getSignature();  
 Method **method** = signature.getMethod();  
 RequestLog log = **method**.getAnnotation(RequestLog.class);  
 String desc = log.operationDesc();  
 String **module** = log.**module**();  
 operLogPO.setModule(module);  
 operLogPO.setOperationDesc("模块:" + module + ",操作行为:" + desc);  
 logger.info("前置通知>>>>>>>>>>>>>>>操作模块：" + **module** + ",操作方法：" + methodTarget + "，操作行为：" + desc + "<<<<<<<<<<<<<<<<");  
 Object result = null;  
 **try** {  
 result = point.proceed();  
 *// 设置请求结果*  
 operLogPO.setResult(JsonUtils.toJson(result));  
 *// 返回通知(操作成功:1,操作失败:2)*  
 operLogPO.setStatus("1");  
 } catch (Throwable e) {  
 operLogPO.setStatus("2");  
 *// 异常通知*  
 throw **new** RuntimeException(e);  
 } finally {  
 *// 后置通知*  
 logger.info("后置通知>>>>>>>>>>>>>>>操作模块：" + **module** + ",操作方法：" + methodTarget + "，操作行为：" + desc + ",操作结果：" + operLogPO.getStatus() + "!(操作成功:1,操作失败:2)<<<<<<<<<<<<<<<<");  
 logRepository.insert(operLogPO);  
 }  
 return result;  
 }  
  
 */\*\*  
 \* 功能：获取IP地址  
 \*  
 \* @param request  
 \* @return  
 \*/*  
 public static String getClientIpAddr(HttpServletRequest request) {  
 String ip = request.getHeader("x-forwarded-for");  
 **if** (ip == null || ip.length() == 0 || UNKNOWN.equalsIgnoreCase(ip)) {  
 ip = request.getHeader("Proxy-Client-IP");  
 }  
 **if** (ip == null || ip.length() == 0 || UNKNOWN.equalsIgnoreCase(ip)) {  
 ip = request.getHeader("WL-Proxy-Client-IP");  
 }  
 **if** (ip == null || ip.length() == 0 || UNKNOWN.equalsIgnoreCase(ip)) {  
 ip = request.getRemoteAddr();  
 }  
 return ip;  
 }  
  
 */\*\*  
 \* 功能：获取浏览器版本  
 \*  
 \* @return  
 \*/*  
 public String getBrownerNo(HttpServletRequest request) {  
 return getNo(request, new String[]{"MSIE", "FIREFOX", "CHROME", "SAFARI", "OPERA"});  
 }  
  
 */\*\*  
 \* 功能：获取操作系统版本  
 \*  
 \* @return  
 \*/*  
 public String getOsNo(HttpServletRequest request) {  
 return getNo(request, new String[]{"WINDOWS NT", "IOS"});  
 }  
  
 */\*\*  
 \* 获取参数  
 \*  
 \* @param request  
 \* @param osNos  
 \* @return  
 \*/*  
 public String getNo(HttpServletRequest request, String[] osNos) {  
 String userAgent = request.getHeader("user-agent");  
 String osNo = "";  
 **if** (userAgent != null) {  
 String str = userAgent.**to**UpperCase();  
 for (int i = 0; i < osNos.length; i++) {  
 **if** (str.indexOf(osNos[i]) > 0) {  
 String str1 = str.substring(str.indexOf(osNos[i]));  
 **if** (str1.indexOf(";") > 0) {  
 osNo = str1.substring(0, str1.indexOf(";"));  
 } **else** {  
 osNo = osNos[i];  
 }  
 break;  
 }  
 }  
 }  
 return osNo;  
 }  
}

最核心部分就是添加 **@Around**注解的方法了，这里是切面的处理中心，我们可以将获取的参数和接口的返回值插入到**Mongodb**中。

### 结果测试

最后我们来测试一下整个过程是否没正确，我们通过**Postman**进行post请求，观察Mongodb中是否有请求日志插入，如果数据插入成功且正确，就说明我们的代码是没问题的，下面我们来看测试结果。